Run-Time Satellite Tele-Communications Call Handling as Dynamic Constraint Satisfaction

Christian Plaunt*
NASA Ames Research Center
Mail Stop 269-2
Moffett Field, CA 94035
(650) 604-2928
plaunt@ptolemy.arc.nasa.gov

Ari K. Jónsson†
NASA Ames Research Center
Mail Stop 269-2
Moffett Field, CA 94035
(650) 604-2799
jonsson@ptolemy.arc.nasa.gov

Jeremy Frank*
NASA Ames Research Center
Mail Stop 269-2
Moffett Field, CA 94035
(650) 604-2524
frank@ptolemy.arc.nasa.gov

Abstract — The next generation of communications satellites may be designed as a fast packet-switched constellation of spacecraft able to withstand substantial bandwidth capacity fluctuations due to causes ranging from unstable weather phenomena to intentional jamming. Scheduling and servicing call requests in such a dynamic environment requires real-time decisions with regard to allocation of resources including bandwidth, call routing and load balancing.

In this paper, we present a general satellite communication scheduling domain, and describe a working implementation of an autonomous system for handling such dynamic scheduling problems. The solution approach is drawn from the area of dynamic constraint satisfaction problems (DCSP), which generalizes these and many other dynamic scheduling problems. We adapt DCSP techniques to the satellite communications domain, in particular solution repair and optimization by gradient-climbing.

These reasoning methods respond to changes in the problem specification by repairing the current solution. As a result, they are anytime algorithms which can trade run-time efficiency for solution quality. This approach supports dynamic call requests; negotiation and fulfillment of prioritized Quality of Service (QoS) contracts; graceful degradation in the presence of dynamic call traffic, changes of priority schemes, and environmental conditions; and optimization of geometrically constrained resources.

TABLE OF CONTENTS

*Caelum Research Corporation
†Research Institute for Advanced Computer Science

1. INTRODUCTION

The current revolution in information technology continually produces new advances in communications capability. One of the critical technologies being closely scrutinized is the application of Asynchronous Transfer Mode (ATM) technology to satellite communications systems. Satellites are limited and expensive communications resources, and ATM technology, through Quality-of-Service (QoS) contracts and statistical multiplexing, offers greater flexibility and resource utilization than existing best-effort and first-come-first-serve systems currently used by some satellite telecommunications technologies.

However, applying ATM to support communication satellites requires innovations beyond standard ATM networks. While ATM offers advanced guarantees and hierarchical resource allocation, frequently ATM does not incorporate priorities in it's guarantee model. One of the goals of this work is to support these domain requirements while increasing the efficiency of resource utilization and supporting dynamic resource allocations. In addition, we must also support geometric constraints and optimizations resulting from satellite beam management.

We use the Planner/Scheduler (PS) and Smart Executive (Exec) subsystems of the Remote Agent (RA) (Bernard *et al.* 1998; Pell *et al.* 1998; Muscettola *et al.* 1998) to implement a new system for satellite telecommunication. The RA will be the first artificial intelligence-based autonomy architecture to reside in the flight processor of a spacecraft (NASA's Deep Space One (DS1)). Similar to other high-level control architectures (Bonasso *et al.* 1997; Wilkins *et al.* 1995; Simmons 1990; Musliner *et al.* 1993), this RA for satellite telecommunications clearly distinguishes between a *deliberative* and a *reactive* layer. In the work reported here, we focus on the reactive layer, the Exec, and its use as a run-time dynamic constraint solver.

We are motivated by the requirements of complex, critical, and highly dynamic satellite tele-communications systems. In this domain, there are several conflicting goals which influence many levels of design choices (for instance, guaranteed

connections versus maximal network throughput, fluctuating bandwidth, conflicting demand patterns, and quality of service). These considerations make this a particularly interesting domain for our exploration. A communication network in this domain must be highly configurable and controllable in order to handle the strategic needs of the user, and also be highly autonomous in order to function efficiently in the potential absence of such control.

Communications Scheduling

The central issue in a QoS communications network is to allocate resources in response to requests for communications. For a concrete example, let us consider a communications network that is based on a small number of multi-beam communication satellites. Each satellite has multiple antennae, each of which can be slewed to an area within the range of the satellite's position. The aim of each antenna determines which ground terminals can use the beam associated with that antenna. This is illustrated in Figure 1. In order to connect a ground station to another ground station, we must find a beam that covers the first ground station and has sufficient uplink capacity, and find a beam that covers the second ground station and has sufficient downlink capacity. Once this has been done, the request can be serviced and the appropriate uplink and downlink capacity associated with this connection.

Given the expense of satellites and the demand for communications abilities, it is not surprising that typically there is demand for more bandwidth than is actually available. This means that connection requests may be rejected and, where very important connections must be made regardless of existing connections, some connections may have to be terminated in order to make more bandwidth available. The simplest option is of course to reject any requests for which bandwidth is not available. However, this solution may reject more requests than necessary, or may reject high priority calls because low priority calls have all available resources. The handling of rejected and terminated connections is typically not an issue in traditional QoS networks. Even assuming that there are no other networks available, such communication requests could simply be requested again, after a suitable delay. In the case of automated systems doing one-way transfers, e.g, image transmission, the sending terminal can even go into batch mode while waiting to get reconnected.

The core of the complexity of the circuit-switched network domain is that there may be more than one route between two terminals. In the satellite example in Figure 1 for instance, multiple beams may cover each of the terminal stations involved. This not only means that we must look at multiple uplink and downlink paths, but also that by moving existing connections between beams or paths, we may find sufficient capacity to handle an incoming connection request which might otherwise not have been accepted. Although this would be significantly simplified by the (unrealistic) assumption that each request needed the same bandwidth, the problem would not go away. Existing connections might still need to migrate to other beams that cover their ground terminal if one or more of the beams associated with a particular connection is moved, for example to cover incoming calls from other areas (see Section 3 and Figure 4 for more details).

The process of moving existing connections, referred to here as *call migration*, interacts with any priority rules that may have been specified for the network. For example, let us imagine a connection request that can only be assigned to a single uplink beam that is at full capacity and that each connection on that beam has higher priority than the new request. It may still be possible to handle the new request by moving one of the higher priority connections to another beam, making room for the new call.

An important concern in the field of communications is response time, i.e., the time from a when a call request is submitted to when it is connected (or denied) should be minimal. This has typically led to rather simple methods for handling conflicts and complications, e.g., simple call rejection when routes with enough capacity cannot be found. However, as the demands on communication networks increase and the types of connections get more and more diverse, the need for more sophisticated, yet efficient, connection request handling systems, becomes ever more pressing.

A Brief Background on ATM

The domain consists of a constellation of spacecraft which act as ATM switches directing and controlling traffic flow from a number of ground telecommunications sources and destinations. Traffic is based on an ATM model with different call *contract types* with the addition of call *priorities*. Contracts ensure a Quality of Service (QoS) so that guarantees can be made a priori about specific types of call connections. At connection setup time, the user must inform the network of both the expected nature of the traffic and the type of QoS contract required. Following are some terms from the ATM literature (see (Varma 1997) for a concise tutorial) we will use in this paper:

- CBR (Constant Bit Rate): Bit rate remains constant over duration of connection; requires delay, jitter and bandwidth guarantees[1].
- VBR (Variable Bit Rate): Intended for supporting bursty data and defined by peak and average rate.
- ABR (Available Bit Rate): Intended for data transmission which do not preserve any timing relationship between source and destination.

In addition calls have *priorities*. Call priorities are designed to ensure that critical calls which need to get through under all circumstances are guaranteed bandwidth capacity while those of lower priority — regardless of contract type — or of a

[1] We distinguish here between different peak and average bandwidth requirements among QoS contracts. E.g., CBR 2 requires roughly twice as much bandwidth as CBR 1.

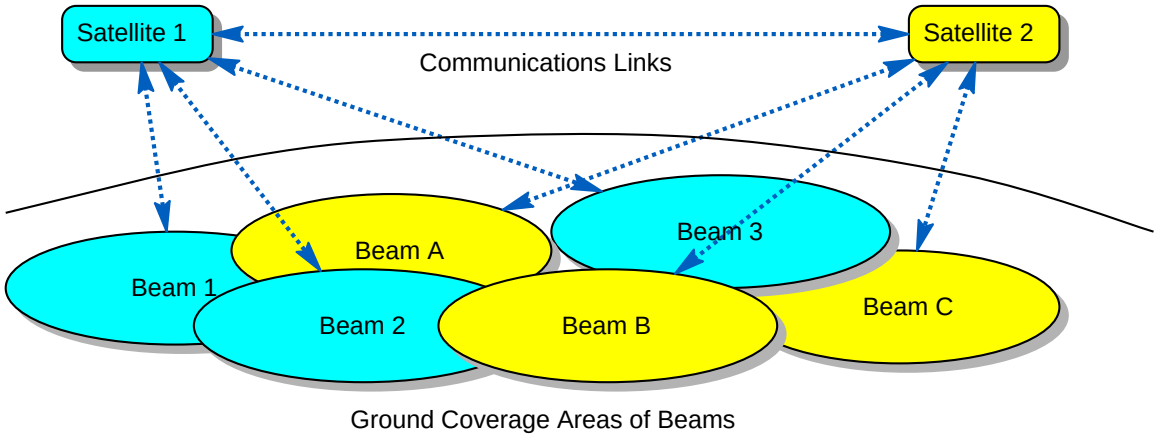

Figure 1: A small network consisting of two satellites, each with three uplink/downlink beams. Coverage areas can overlap, beams can be slewed to various areas. Any two ground stations can communicate via any combination of beams and satellites.

non-critical nature are allocated bandwidth on an as available basis. Such calls may be of any contract type, depending on the nature of the call (voice, video, data, etc.). Less critical calls might request an "expensive" contract (e.g. CBR), but also be willing to accept a less expensive contract (e.g. ABR) if that is the best contract available. While call priorities are not part of traditional ATM QoS models, they can be viewed as an extension of ATM QoS; that is, contracts are guaranteed as long as higher priority calls do not require those resources.

Currently, such communication is managed by restricting the identity, time, and bandwidth allocations of people and equipment that can use the system to communicate. Multiple high priority channels are reserved just in case an important message needs to be sent. In this approach not only is the complete bandwidth allocation preallocated as a "pipe" (i.e. once allocated the resources are completely tied to the user), but dynamic request allocations can only be accepted if the request is of a high enough priority to preempt an ongoing call when there is insufficient capacity. Needless to say, this is a highly suboptimal approach, especially in circumstances where frequently a large amount of bandwidth is needed on demand and where no accurate predictions can be made a priori. Adding priorities and call termination to a ATM QoS network makes this work unique.

A Brief Background on QoS Networks

There is a large and growing body of work on QoS routing. Apostolopoulos *et al.* (1998) define Quality of Service routing as "the process of selecting the path to be used by the packets of a flow based on its QoS requirements, e.g. bandwidth or delay." In the call servicing domain, this can be reinterpreted slightly, to be the allocation of resources for the routing of a call based on its QoS requirements.

QoS routing is meant to improve the service received by users of the network in whatever required form, be it enhanced network throughput, minimized network response time, minimum variability in network performance, and so on. It is also designed to enhance network revenue; each completed call results in some reward, as in Marbach *et al.* (1998).

A QoS scheme consists of several components: *call admission control*, *route selection*, *update policies*, and *call management*. In our scenario, there is no update policy; each ground station is assumed to know the status of the beam, and there is no need to distribute information to remote parts of the network. This eliminates uncertainty about available bandwidth and protocol overhead. We discuss the remaining aspects below.

Route Selection — Circuit switched networks reserve routes in advance. One protocol to reserve routes is RSVP, which manages resources in support of QoS routing. RSVP messages are sent on the path traffic will take, notifying switches along that route of the resources which the traffic requires. In an ATM setting, this assumes the route is allocated beforehand (Berson 1996).

Call Admission Control — Call admission control (CAC) is the process of deciding whether or not to accept a call. Greedily accepting all calls may result in suboptimal performance as high priority calls are blocked by low priority calls. Marbach *et al.* (1998) and Tong & Brown (1998) discuss reinforcement learning approaches to devising CAC policies to maximize expected reward in a QoS network. Apostolopoulos *et al.* (1998) investigate a CAC policy in conjunction with various updating schemes for a path selection procedure in a QoS setting.

Call Management — Frequently, as in Apostolopoulos *et al.* (1998), QoS routing involves selecting a path for a flow or call and maintaining that flow until the call completes; hence call management is limited to registering the beginning or completion of a call. In cellular telephone networks and other mobile communication networks employing protocols such as Mobile IP, call migration does occur. To our knowledge there is no system implementing QoS routing on such networks, however in principle it could be done. In cell phone

network, QoS requirements could dictate which of overlapping base stations calls are handed off to.

2. Modeling the Problem As Constraint Satisfaction

Communications scheduling problems are instances of *constraint satisfaction problems*. These problems are characterized by a set of constraints on a finite set of variables, each having a finite domain of values. Since many real-world problems can in fact be described as constraint satisfaction problems, this area has received a great deal of attention in recent years. This interest in constraint satisfaction has produced a number of important advances, including methods for solving realistically sized problems quite efficiently. In fact, a number of real-world problems are currently being solved in industrial applications, using the techniques developed for constraint satisfaction.

At a first glance, it may seem unintuitive to look at a more general class of problems when dealing with time critical real-time reasoning. However, the literature on constraint satisfaction includes a number of efficient techniques that can be applied in domains such as real-time decision making. Furthermore, the field is well established and formal, which gives us clear and developed definitions, and more importantly, a large library of existing results. This combination makes constraint satisfaction a powerful tool for helping to solve problems such as communications scheduling problems.

Constraint Satisfaction Problems

A constraint satisfaction problem (CSP) is a set of variables, each taking values from a finite domain, along with a set of constraints on which combinations of variable assignments are allowed. A solution to a CSP is a listing of variable-value pairs, such that each variable is mentioned exactly once, and that these assignments satisfy all the constraints. To specify this formally:

A *constraint satisfaction problem* is a triple $C = (X, V, K)$, where:

1. $X = \{x_1, \ldots, x_n\}$ is a set of variables
2. $V = V_{x_i} \mid i \in \{1, \ldots, n\}$, where each V_{x_i} is a finite set of possible values of x_i.
3. K is a set of constraints (Y_j, R_j), where $Y_j = \{x_{i_1}, \ldots, x_{i_k}\} \subseteq X$ and $R_j \subseteq \prod_{\nu=1}^{k} V_{x_{i_\nu}}$.

A *solution* to a constraint satisfaction problem $C = (X, V, K)$, where $X = \{x_1, \ldots, x_n\}$, is an n-tuple $(v_{x_1}, \ldots, v_{x_n})$, such that:

1. $v_{x_k} \in V_{x_k}$ for $k = 1, \ldots, n$, and
2. For any $(Y, R) \in K$ with $Y = \{x_{i_1}, \ldots, x_{i_k}\}$, we have $(v_{x_{i_1}}, \ldots, v_{x_{i_k}}) \in R$.

Many methods are available for solving constraint satisfaction problems. There are two major categories of solution techniques. *Complete* methods are guaranteed either to find a valid assignment of values to variables or prove that no such assignment exists[2]. The problem is that in the worst case, complete methods may require run-time that grows exponentially with the number of variables to find a solution. So, although such methods frequently do exhibit good performance, and can guarantee a correct answer for all inputs, solutions can often be found faster by using methods that may not be complete. Such *incomplete* methods cannot prove that a CSP has no satisfying assignment, but, given sufficient time, they can find satisfying assignments with high probability, if solutions exist. These incomplete algorithms have gained popularity in recent years, due to their simplicity, speed and observed effectiveness at solving certain types of problems.

Constraint Optimization Problems

When it comes to real world constraint applications, the issue is typically not just finding a solution, but finding a good one. For many problem domains, such as scheduling, this can significantly add to the complexity of solving the problem. It is, for example, trivial to find a valid job shop schedule, but it is in general intractable to find one of minimal length. The good news is that when it comes to real-time systems, one can often take advantage of how easy it is to find a valid, but not necessarily optimal solution to the typical optimization problem.

To formalize this concept, we define a *constraint optimization problem* as a pair (C, g) where C is a constraint satisfaction problem and g is a function that maps every valid solution[3] to C into $\mathbb{R}$. An equivalent definition of the optimization criteria is that it specifies a partial ordering of solutions. In some cases, such as when dealing with priorities, partial preference ordering is more intuitive than a real function. Either way, we can view the optimization function as a gradient within the subset of valid solutions. The goal of constraint optimization is to find a valid solution that is a global maxima within the preference gradient.

As an example of a constraint optimization problem, let us consider a simple scheduling problem consisting of two low-capacity beams and a few call requests (disregarding geographic constraints for simplicity). The call requests are then variables that must be assigned beam allocations (or rejected), and the constraints limit the overall bandwidth requirements for all calls assigned to a beam. Finding a satisfactory assignment of calls to beam 1, beam 2 or rejection, is a constraint satisfaction problem. If we now specify a preference among

[2] This does not mean that complete methods necessarily look at all possible assignments. Most assignments are analyzed and eliminated by other means of proving them invalid.

[3] The *valid solutions* of a CSP are all the assignments which satisfy the constraints.

solutions, or equivalently, specify an optimization function on the set of valid assignments, we have a constraint optimization problem. In other words, the constraint satisfaction part defines a set of valid call assignments, and the preference functions defines an evaluation of each solution, based on call priorities.

Dynamic Constraint Satisfaction Problems

Another factor in real-world problems is dynamism; the world is constantly changing, and often we are presented with problem instances that evolve over time, rather than remaining static. One could simply view and solve each variation separately, but in certain situations that may not be possible, given real-time response requirements. Furthermore, we will usually find that it is more efficient to view them as related and solve them as a sequence of connected problems. Such sequences of changing constraint satisfaction problems are called dynamic constraint satisfaction problems.

To formalize this notion, let $C = (X, V, K)$ be a constraint satisfaction problem. Then, any problem of the form $C' = (X', V', K')$, $C' \neq C$, such that $X' \supseteq X$, $V'_x \subseteq V_x$ for each $x \in X$ and $K' \subseteq K$, is a *restriction* of C. And any problem of the form $C' = (X', V', K')$, $C' \neq C$, such that $X' \subseteq X$, $V'_x \supseteq V_x$ for each $x \in X$ and $K' \supseteq K$, is a *relaxation* of C. A *dynamic constraint satisfaction problem* is a sequence of constraint satisfaction problems $C_0, C_1, \ldots$, such that each problem C_i is either a *restriction* or a *relaxation* of C_{i-1}. This definition is consistent with Dechter & Dechter (1988) and Verfaillie & Schiex (1994).

Not surprisingly, it is relatively straightforward to generalize the idea of dynamic constraint satisfaction to dynamic optimization problems. Formally, a *dynamic constraint optimization* problem is a sequence of optimization problems, such that each entry is a relaxation or a restriction of the previous problem. This means that the optimization function remains unchanged throughout, but the set of variables, domains and constraints may change.

Hill Climbing

Hill climbing is one of the most popular approaches to solving constraint satisfaction problems in an incomplete fashion[4]. The basic idea is to define an evaluation criteria for each valid solution (or invalid full assignments), such that improving the evaluation leads to better (or closer to valid) solutions. Needless to say, choosing this evaluation function is a critical aspect of successful hill-climbing search. The other main aspects of hill-climbing are the operation that maps one solution candidate to a set of possible successors, and the selection process for picking the successor.

To clarify the notion of hill-climbing, let us again consider a simplified call allocation problem. A trivial, valid solution to this problem consists of rejecting all the call requests. This is the worst valid solution, given the optimization criteria, but it is nonetheless valid. To define a hill-climbing algorithm, we can therefore use the optimization function itself to evaluate our candidate solutions, and use randomization to select among ties. To finish specifying the hill-climbing search method, we must define an operation that maps one candidate solution to a set of other candidates. A possibility is to select a call that is currently rejected, and try assigning it to a beam, possibly displacing lower priority calls. The hill-climbing mechanism then picks the call-beam combination that maximizes the optimization function.

To take a concrete example, assume we have assigned a call of priority 5 requiring a bandwidth of 2, to a single given beam with capacity of 4 units[5]. Two calls are currently rejected, one with priority 3 and bandwidth requirement of 3, and the other with priority 7 and bandwidth requirement of 1. This current solution could be described as $(\{C_{5,2}\}, \{C_{3,3}, C_{7,1}\})$, with the first set being calls assigned to the beam and the second set consisting of rejected calls.

Let us then choose a simple optimization function, which sums up $(10 - p) \cdot b$, where p is the priority (1 highest, 8 lowest) and b is the bandwidth used. Our current solution then evaluates to $(10 - 5) \cdot 2 = 10$. Our successor function might then give the following options:

$$(\{C_{3,3}\}, \{C_{5,2}, C_{7,1}\})$$

which evaluates to 21, and

$$(\{C_{5,2}, C_{7,1}\}, \{C_{3,3}\})$$

which evaluates to 13. We therefore pick the first candidate as the new current solution.

A second hill-climbing iteration would then result in

$$(\{C_{3,3}, C_{7,1}\}, \{C_{5,2}\})$$

which is indeed an optimal solution at 24.

Unfortunately, one cannot expect hill-climbing algorithms to find optimal (or even near-optimal) solutions that easily for real problems. In fact, it is invariably intractable to find optimal solutions to real problems. However, hill-climbing methods have the distinct advantage that they can provide a valid solution at any time-point. This makes the technique very suitable for systems that must perform with real-time guarantees. An added bonus is that the more time the hill-climbing process is given, the better the solution will typically be. It is therefore not surprising that our approach to solving the runtime satellite scheduling problem, is based on hill-climbing. We now turn our attention to the exact hill-climbing algorithm we developed.

[4] The two best known successful hill-climbing techniques are GSAT (Selman *et al.* 1992) which is used to solve SAT problems, and Min-Conflicts (Minton *et al.* 1991) which is used to solve general constraint satisfaction problems.

[5] To clarify the issues in specifying a hill-climbing algorithm, this sample problem is greatly simplified. It is not representative of the actual problem solved by the full algorithm, which is described below.

3. Dynamic Call Routing and Management

The run-time system developed and described in this paper is the reactive real-time subset of the Remote Agent satellite tele-communications architecture shown in Figure 2 (Plaunt *et al.* 1998). The principal components of this architecture — the Call Admission Controller, the Load Balancer, the Contract Manager and the Priority Table Managers — implement the domain specific DCSP engine which is the subject of this paper. This architecture is based on the run-time components of the Remote Agent (Pell *et al.* 1998) plus the unshaded domain specific components shown in this figure.

In this application, the run-time execution system's objective is to enforce a priority based communication policy in a variety of environmental and network loading situations. In the current system, the communication policy of interest is (1) to service all dynamic communication requests in highest priority first order until either all requests are serviced or bandwidth capacity is reached; and (2) when bandwidth capacity is reached (or exceeded), to migrate or to shed communication allocations in lowest first priority order until bandwidth usage is back within capacity. That is, the major functions of the implemented algorithm are those concerned with the constraints on call admission and load balancing.

Priority Management

Policy enforcement is based on a system of call priorities. That is, when enforcing these policies, whenever a conflict arises either during the call admission process or in the management of the current calls in progress (load balancing), the system consults dynamic priority table managers to determine which call or calls will be accepted, denied, migrated, or shed. An example of such a table which determines the "priority rank" of a particular call is shown in Figure 3.

Call Acceptance

All arriving call requests are handled by the Call Admission Controller (CAC). The CAC admits call requests to the system based on the call's priority rank and system resource availability. When a new call request is received, the CAC proceeds in the following steps:

1. attempt to find a route for the call, and if one is found:
2. find a solution to the new DCSP which includes the new call request (i.e. run load balancer):
3. if the new call request is part of the new solution (i.e. survives load balancing), accept it, otherwise deny it.

Finding a route for a call involves first finding both uplink and downlink beams with coverage for both the call's origin and destination (see Figure 4 for a conceptual view of beam coverage). Note that this is done without checking for *capacity* on either the uplink or downlink beams. The reason for this is that if either of the beams are at or close to capacity, it may be necessary for the new call to bounce one or more lower priority existing calls in order to be accepted (see next sub).

This is exactly what the second step of call acceptance does. That is, given the new call request *and* all of the currently in process calls, the system finds a new solution which satisfies the priority, bandwidth and coverage constraints. The solution must be no worse than the current solution, and must be derivable from the current solution with very little search.

Once a new solution is found, if the new call request is part of that solution, then the call is accepted, admitted to the system, all changes required to realize the new solution are taken, and the call becomes one of those managed as part of load balancing. If the new request is *not* part of the new solution (which can happen when one of the beams in at or near capacity and the new request is of low priority), the call is denied and no resources are allocated to it.

In terms of the DCSP framework, the addition of a new request constitutes a restriction of the CSP. The solver attempts to repair the previous solution by extending the assignment to the new variables, then conducting a form of hill-climbing search to improve the assignment. If this can be done without terminating a connection, then no further modification of the problem instance occurs. If an existing call must be terminated, or if the new call request must be rejected, this constitutes a relaxation of the CSP. Further, note that if the new call was rejected then no new solution could be found.

Load Balancing

The purpose of load balancing in this application is to optimize bandwidth usage within the capacity available on the network and the coverage provided by the uplink and downlink beams in the current system configuration.

The load balancing machinery is invoked when deciding whether to accept a call and whenever the call load on the system exceeds the system resources to support such calls. This can happen:

1. when a new call request would put one or more of the beams on its route over capacity,
2. when a beam's capacity drops below its current usage (e.g. due to environmental interference, component failures, jamming, etc.),
3. when a beam moves away from a coverage area and can no longer provide coverage for either the uplink or downlink of one or more of the connections on that beam.

When a beam moves, some calls associated with that beam may no longer have coverage. For example, if the uplink illustrated in Figure 4 Configuration A is on Beam 2 at some

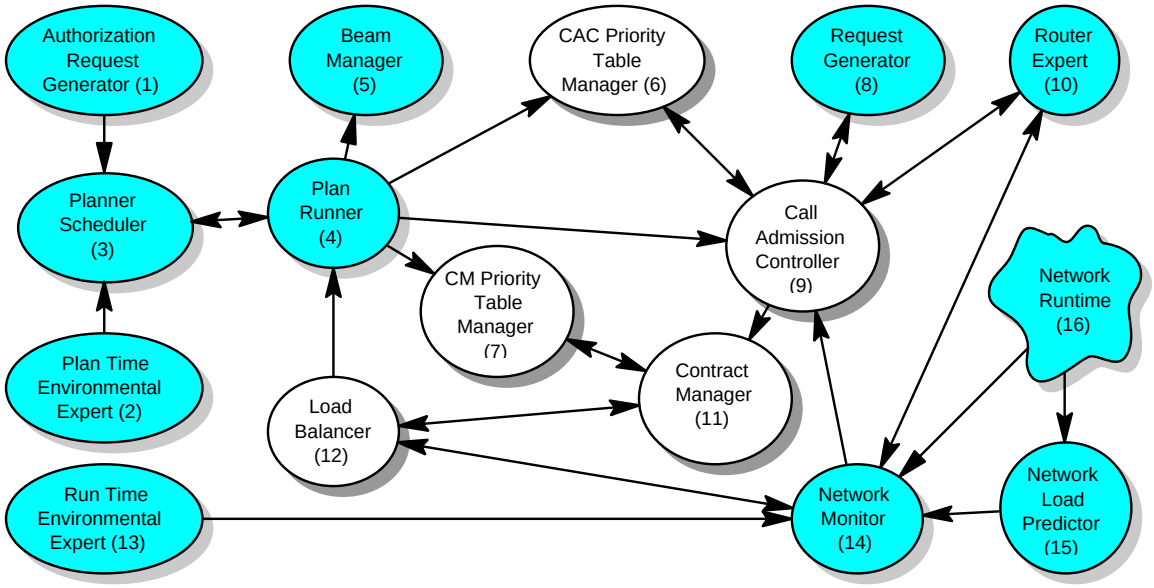

Figure 2: The application architecture for a modular, high level ATM network controller. The run-time components which implement the dynamic constraint solver for this domain — the Call Admission Controller, Load Balancer, Contract Manager and the Priority Table Managers — are unshaded.

Rank	1	2	3	4	5	6	7	8
Contract Priority	CBR 1 high	CBR 2 high	VBR 1 high	CBR 1 medium	CBR 2 medium	ABR 1 high	ABR 2 high	ABR 1 medium

Figure 3: An example of a priority table. Priority rank is determined as a function of a call's assigned priority and QoS contract. That is, the number one ranked calls are "high" priority CBR 1 calls, the second rank are "high" priority CBR 2 calls, and so on.

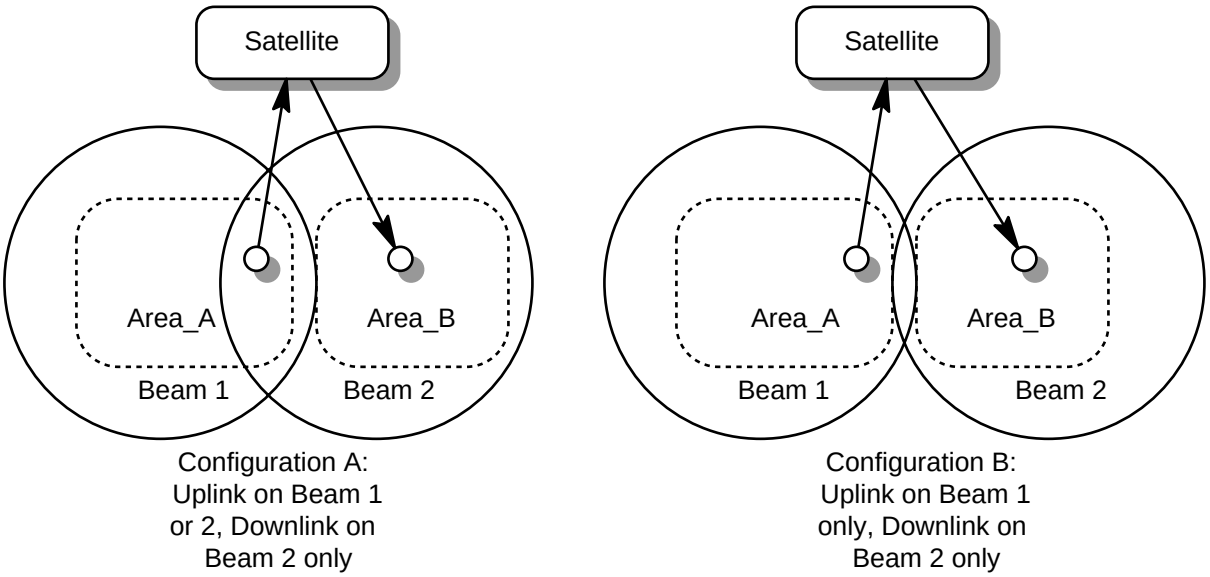

Figure 4: An illustration of two possible situations which arise in call routing and migration. Uplink and downlink beams may or may not overlap (Configuration A and B respectively). Calls may be un-routable, or routable via one or more routes. Beams move and change coverage areas over time. Areas may be covered by zero, one, or more beams.

point, and that beam is then moved as in Figure 4 Configuration B, then that uplink must either be moved from Beam 2 to Beam 1 or the call must be dropped. However, before the uplink can be moved, Beam 1 has to be checked for the required capacity for a call of this particular priority rank at this time. If it has the capacity then the uplink can be moved, otherwise the call must be dropped.

Load balancing must be fast and must disrupt the system (and hence the DCSP solution) as little as possible when invoked. That is, it is acceptable to trade some quality in the new solution for more efficiency in the search for, and realization of, a solution.

Since the load balancer is only invoked when system usage would exceed capacity (or a given threshold), its primary job is to migrate or shed the lowest priority calls in the system in order to provide continuous ("circuit switched") service to the higher priority calls.

Call Shedding and Migration — When the available bandwidth or coverage on a particular beam can no longer support all of its uplink or downlink usage, some calls must either be re-routed or dropped in order to keep the beam's usage within its capacity. As a matter of policy, this task must be carried out such that:

1. calls are only shed as a last resort when they can't be migrated or deferred (e.g. an ABR call may have its bit rate turned down in order save bandwidth);
2. although migrating one call to make room for another is recursive (i.e. the call being migrated may in turn force another to be moved), since only calls of strictly lower priority can be bumped, the search terminates very quickly.

The implemented load balancing algorithm is:

1. if the uplink on this beam is over capacity:
   (a) find the lowest priority call on this uplink beam,
   (b) make a list of alternate beams which have the required uplink coverage for this call, sorted in most-capacity-first order,
   (c) for each beam in this list of alternate beams, try to migrate this lowest priority call to this alternate beam;
      i. if the beam has capacity, move this call and move on to the next,
      ii. if the beam does not have sufficient capacity *and* a set of calls of sufficient capacity with strictly *lower* priority can be found on the alternate beams, all of these calls are migrated (or terminated) and the new call is moved onto this beam,
   (d) if the beam does not have capacity *and* no capacity can be reclaimed on it, this call is terminated and its resource are released;
2. now do the same thing for the downlink of this beam if it is over capacity (which it may not be — migrating or shedding the uplink may already have cleared enough bandwidth if the uplink and downlink beams were shared).

The algorithm is designed such that the smallest number of calls possible are migrated or shed. That is, only one call at a time is considered for migration or shedding, and before any work on either end of that call (uplink or downlink) is actually done, the beam is checked to be sure the work indeed needs to be done. This happens because the bandwidth for any call must be accounted for separately on its uplink and downlink path. For example, when a call is shed, both its uplink and downlink resources are returned. If the uplink and downlink were on the same beam, then twice as much bandwidth as expected is recovered. If the uplink and downlink are different beams, than both beams reclaim the bandwidth equal the bandwidth used by to that particular call.

Discussion

Figure 5 shows a graph of bandwidth usage by priority over time on one of the downlink beams during a test run of 7769 dynamic call requests over a period of 33 minutes (an average of about 3.9 calls per second). Note that the highest priority calls (those with a rank of "1") are graphed at the bottom, with successive ranks of lower priority calls (ranks 2–8) in increasing order above. The top-most line graphs the total available bandwidth capacity of the beam during the run. This scenario of multiple stepwise decreases in bandwidth followed by a similar set of increases in bandwidth over time is intended to illustrate the reactivity of the system, not to represent a typical operational situation.

This figure shows that as the capacity of this beam changes, calls are only affected in lowest-first priority order. That is, high priority calls behave as though they are in a "circuit switched" pipe, and low priority calls fill in the remaining available bandwidth. However, since the (packet switched) virtual "circuits" are dynamic, there is little or no wasted bandwidth (in contrast to a real circuit which when in use (or reserved), uses all of its allocated bandwidth *regardless* of how many slots of data are actually sent via the switch). In addition, as many priorities as possible are provided this "circuit switched" behavior within the available bandwidth.

When checking to see if a call can be migrated from one beam to another, it is allowed to "bump" the lowest priority call (or calls) on the target beam to make room. The bumped call (or calls) may in turn recursively try to migrate to an alternate beam. However, calls to be bumped must be both the lowest priority on the beam of interest *and* be of a *lower* priority rank than the call which is being migrated. Assume that all

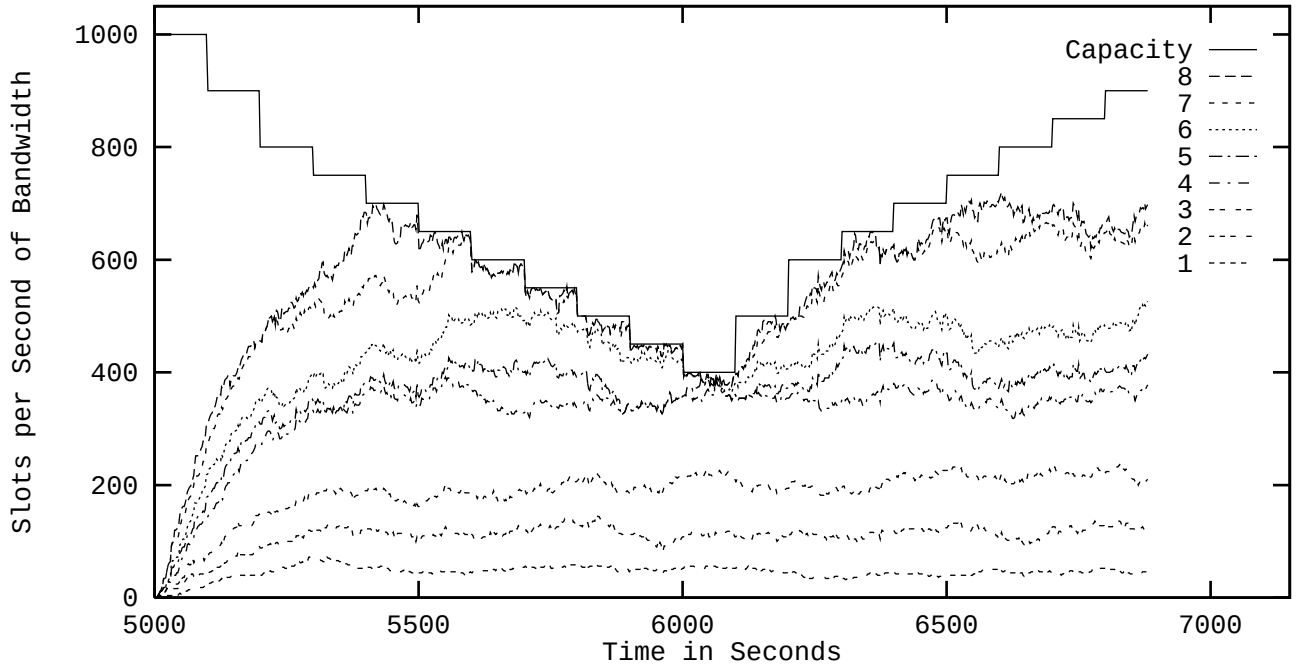

Figure 5: A graph of bandwidth capacity and usage by priority over time on a downlink beam. The highest priority calls (1) are at the bottom, with successive lower priority call in increasing order above (2-8), total available bandwidth capacity at the top.

calls have the same capacity requirements; then migration can only displace a single call. Further, even in the worst case, the restriction that the displaced call must be the lowest priority call on the alternate beam means that after a call has been displaced from each beam, the call being displaced on the last beam is the *lowest* priority call in the system.

We can place an upper bound on the number of migrations performed in each step. Suppose that the number of priorities available is p and the number of beams the satellite supports is b. If $p > b$, then in the worst case migration requires $O(b^2)$ total (recursive) migrations because once the lowest priority call has been bumped from every beam, the call can't migrate anywhere. If $p < b$ then in the worst case migration requires $O(pb)$ total (recursive) migrations because each migration must strictly reduce the priority of the call being migrated. In practice, this number is much smaller because there are typically a small number of beams, the lowest priority is reached very quickly, any calls which are disconnected clear bandwidth on both uplink and downlink beams, and so on. When relaxing the assumption that all calls have the same capacity, this solution still holds, but the complexity now depends on the number of calls which need to be displaced in order to make room for the new call. Finally, note that since there is only a small number of beams and priorities, this upper bound gives us the real-time guarantees we wanted.

4. Conclusions and Future Work

In this paper, we introduced a real world problem domain that can be viewed as a dynamic constraint optimization problem with soft constraints. Communications satellites of the future may be designed as a fast packet-switched constellation of spacecraft able to withstand substantial bandwidth capacity fluctuation ranging from unstable weather phenomena to intentional jamming of communication. Scheduling and servicing call requests in such a dynamic environment requires real time decisions with regard to allocation of bandwidth, call routing, load balancing, call prioritization, and so on. We have explored viewing this as an instance of a dynamic constraint satisfaction problem.

After introducing this dynamic call routing problem, we reformulated it as a constraint satisfaction problem with soft constraints. We then provided a set of hill-climbing methods for handling this problem. Based on a fast, single-pass solution repair mechanism, we showed that the methods are capable of performing the functionality required within the specified time windows.

Finally, we discussed the implementation of the concepts in the form of a domain specific software artifact built from the components of the Remote Agent. This run-time reactive execution system enforces a set of communication policy constraints discussed above in a satellite tele-communications domain consisting of a single satellite with two beams. We showed that the system can provided traditional "circuit switched" like guarantees for hight priority connections, and exploit packet switching in order to optimize bandwidth capacity usage at the same time.

There are two particular comparisons we plan to make in order to quantify the effectiveness of the currently implemented

system. First, we plan to compare the dynamic throughput (in terms of slots per second) of the current system with a circuit switched system. The purpose here is the see if the packed switched approach in fact does provide the kind of gains it appears to. Second, we plan to compare the current system to a series of "cost-no-object" DCSP solutions in order to quantify its optimality.

REFERENCES

Apostolopoulos, G., R. Guérin, S. Kamat, & S. Tripathi (1998). Quality of service based routing: A performance perspective. In *Proceedings of the ACM SIGCOMM*, pages 17–28. ACM.

Bernard, D. E., G. A. Dorais, C. Fry, E. B. Gamble, B. Kanefsky, J. Kurien, W. Millar, N. Muscettola, P. P. Nayak, B. Pell, K. Rajan, N. Rouquette, B. Smith, & B. C. Williams (1998). Design of the remote agent experiment for spacecraft autonomy. In *Proceedings of the IEEE Aerospace Conference*, Snowmass, CO. IEEE.

Berson, S. (1996). Classical RSVP and IP over ATM. INET.

Bonasso, R. P., D. Kortenkamp, D. Miller, & M. Slack (1997). Experiences with an architecture for intelligent, reactive agents. *JETAI*, 9(1).

Dechter, R. & A. Dechter (1988). Belief maintenance in dynamic constraint networks. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 37–42, Palo Alto, CA. Morgan Kaufmann.

Getoor, L., G. Ottson, M. Fromherz, & B. Carlson (1997). Effective redundant constraints for online scheduling. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 302–307.

Marbach, P., O. Mihatsch, M. Schulte, & J. Tsisiklis (1998). Admission control and routing in integrated service networks. In *Advances in Neural Information Processing Systems (10): Proceedings of the 1997 Conference*, Cambridge, MA. MIT Press.

Minton, S., Johnson, M., Phillips, A. & Laird, P. (1991). Solving Large Scale Constraint Satisfaction and Scheduling Problems Using A Heuristic Repair Method. In *Automated Reasoning*, pages 17-24.

Muscettola, N., P. P. Nayak, B. Pell, & B. C. Williams (1998). Remote Agent: To boldly go where no AI system has gone before. *Artificial Intelligence*, 103(1/2). To Appear.

Musliner, D., E. Durfee, & K. Shin (1993). Circa: A cooperative, intelligent, real-time control architecture. *IEEE Transaction on Systems, Man, and Cybernectics*, 23(6).

Pell, B., D. E. Bernard, S. A. Chen, E. Gat, N. Muscettola, P. P. Nayak, M. D. Wagner, & B. C. Williams (1998). An autonomous spacecraft agent prototype. *Autnonmous Robots*, 5(1).

Plaunt, C., R. Kanna, B. Pell, & N. Muscettola (1998). Integrated planning and execution for satellite telecommunications. In *AAAI 1998 Fall Symposium Series: Integrated Planning for Autonomous Agent Architectures*. AAAI.

Selman, B., Levesque, H. & Mitchell, D. (1992). A New Method For Solving Hard Satisfiability Problems. In *Proceedings of the* 9^{th} *National Conference on Aritificial Intelligence*, pages 440-446. AAAI Press.

Simmons, R. (1990). An architecture for coordinating planning, sensing and action. In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 292–297, San Mateo, CA. Morgan Kaufmann.

Tong, H. & T. Brown (1998). Optimizing admission control and routing wile ensuring quality of service in multimedia networks via reinforcement learning. Submitted to IEEE Infocomm.

Varma, A. (1997). Traffic management in ATM networks. In *Proceedings of MILCOM 1997*.

Verfaillie, G. & T. Schiex (1994). Solution reuse in dynamic constraint satisfaction problems. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 307–312, Cambridge, MA. MIT Press.

Wilkins, D. E., K. L. Myers, & J. D. Lowrance (1995). Planning and reacting in uncertain and dynamic environments. *JETAI*, 7(1):197–227.

Christian Plaunt *is a member of the Computational Sciences Division at the NASA Ames Research Center, where his research is focused on intelligent agents and execution systems systems for spacecraft and space-based instruments. His research interests also include intelligent information selection systems, contrapuntal music, the lisp family of languages and observed trials. He holds a Ph.D. from U.C. Berkeley.*

Ari K. Jónsson *is a Research Scientist at NASA Ames Research Center. His research efforts have been focused on efficient constraint satisfaction techniques, including search strategies, propagation algorithms and the use of procedural methods. Currently, he is working on constraint reasoning and planning in complex domains. He has a Ph.D. from Stanford University.*

Jeremy Frank *is a Research Scientist at NASA Ames Research Center. His research interests include machine learning and local search for combinatorial optimization problems. He has worked in the areas of computer network architecture design, computer network protocol design, scheduling, and Boolean satisfiability. He has a Ph.D. from U.C. Davis.*